



Daffodil
International
University

Lab Reports

Course code: CSE422

Course Title: Computer Graphics Lab

Submitted to:

Deawan Rakin Ahamed Remal

Lecturer, Department of CSE

Daffodil International University

Submitted by:

ISHAN AHMAD

203-15-14521

57-B

Department of CSE

Daffodil International University

Submission Date: 16th November, 2023

Experiment No. 1

Experiment Title: An OpenGL program to draw a straight-line using glut

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int x1, y1, x2, y2;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

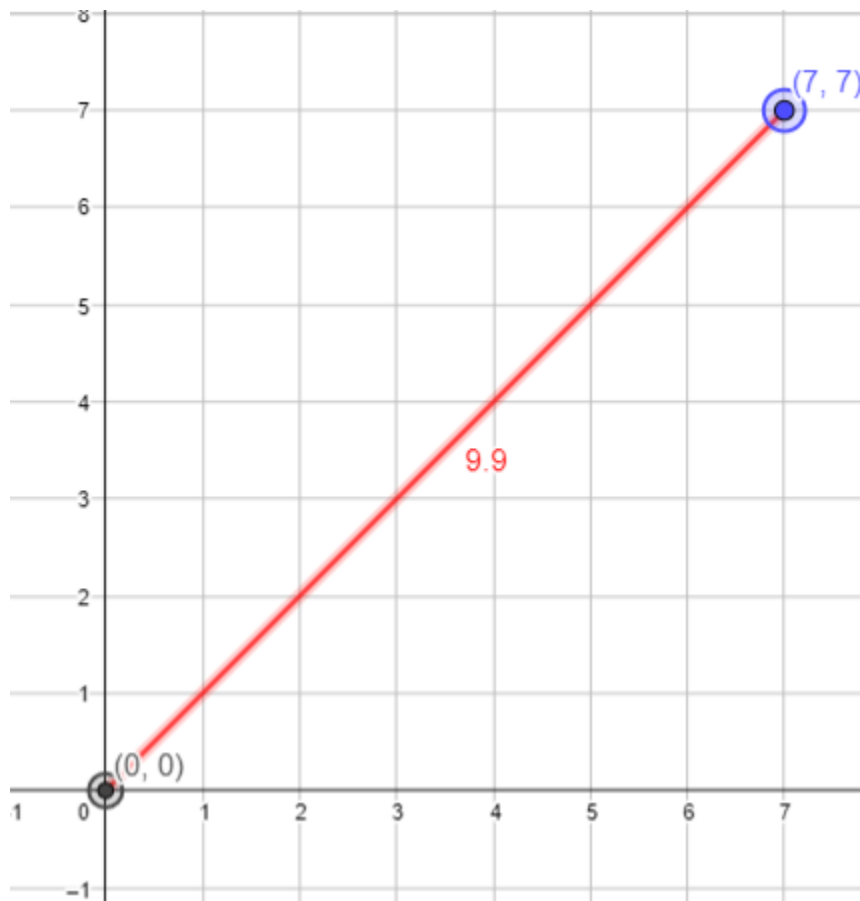
void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_LINES);
        glVertex2i(x1, y1);
        glVertex2i(x2, y2);
    glEnd();

    glFlush();
}
```

```
}  
int main(int argc, char* argv[])  
{  
  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  
  
    glutInitWindowPosition(1200, 100);  
    glutInitWindowSize(700, 700);  
    glutCreateWindow("Straight Line");  
  
    init();  
    glutDisplayFunc(drawShapes);  
  
    cout << "Enter value of starting and end points " << endl;  
  
    cout << "x1: ";  
    cin >> x1;  
    cout << endl;  
    cout << "y1: ";  
    cin >> y1;  
    cout << endl;  
  
    cout << "x2: ";  
    cin >> x2;  
    cout << endl;  
    cout << "y2: ";  
    cin >> y2;  
    cout << endl;  
  
    glutMainLoop();  
  
    return 0;  
}
```

Graph:

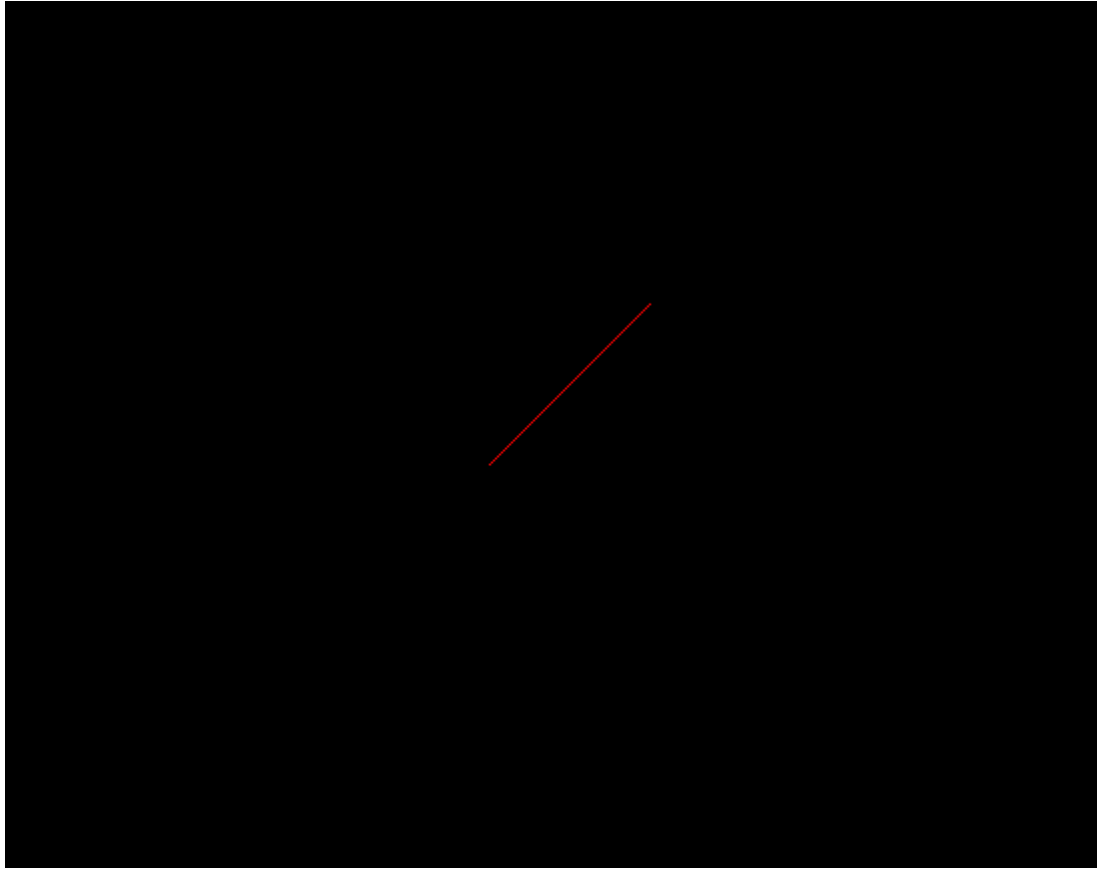


Input:

```
Enter value of starting and end points
x1: 0
y1: 0
x2: 70
y2: 70
```

Input data = graph data * 10

Output:



Discussion:

This C++ program uses the OpenGL library to draw a straight red line on a black background. The `init` function sets up the viewing area, and `drawShapes` draws the line based on user-inputted starting and ending points. The program prompts the user to input the coordinates of two points (x_1, y_1) and (x_2, y_2) to determine the line's position.

Experiment No. 2

Experiment Title: An OpenGL program to draw a quad shape and a triangle

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(0.0, 600.0, 0.0, 600.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_QUADS);
        glVertex2i(70, 170);
        glVertex2i(160, 170);
        glVertex2i(190, 60);
        glVertex2i(50, 90);
    glEnd();

    glColor3f(1, 0, 0);
    glBegin(GL_TRIANGLES);
```

```
        glVertex2i(330, 370);
        glVertex2i(270, 260);
        glVertex2i(180, 260);
    glEnd();

glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

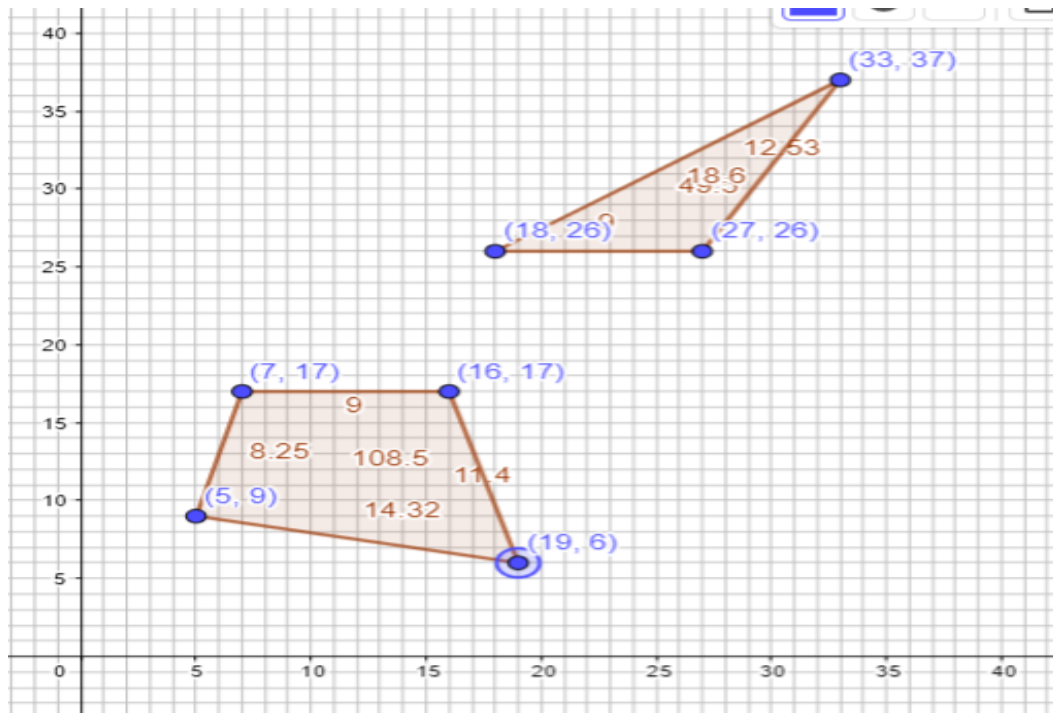
    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("Quad Shape and Triangle Shape");

    init();
    glutDisplayFunc(drawShapes);

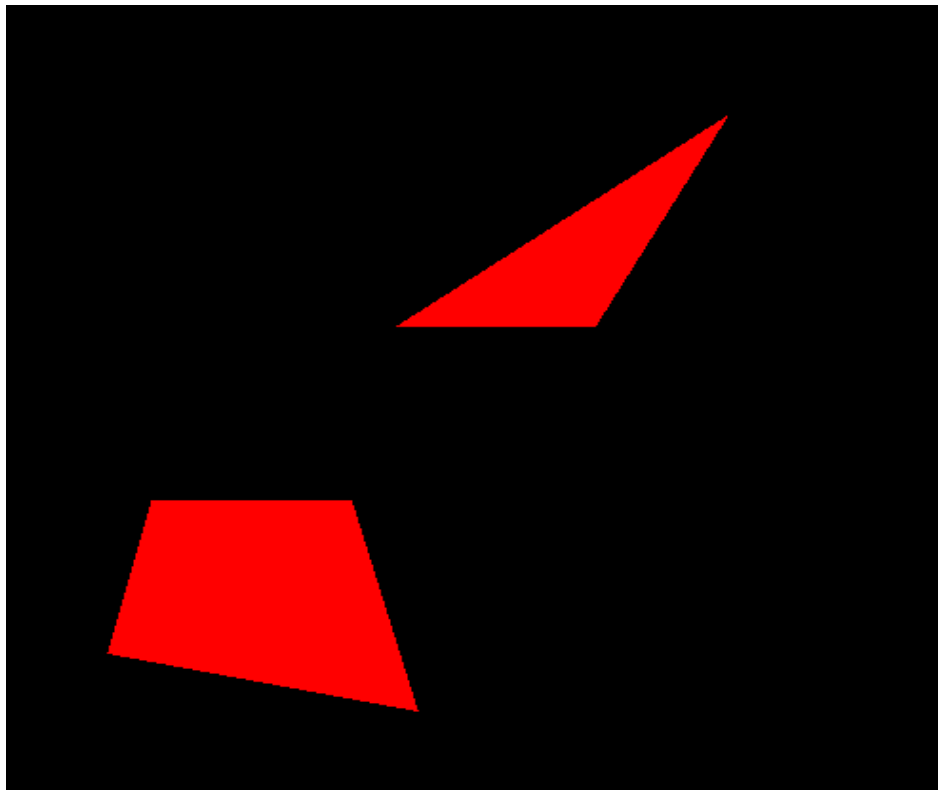
    glutMainLoop();

    return 0;
}
```

Graph:



Output: (Output data = graph data * 10)



Discussion:

This C++ program uses OpenGL to draw a red quadrilateral (rectangle) and a red triangle on a black background. The `init` function sets up the viewing area, and `drawShapes` specifies the coordinates of the vertices for both shapes. The quadrilateral has vertices at (70, 170), (160, 170), (190, 60), and (50, 90). The triangle has vertices at (330, 370), (270, 260), and (180, 260). The program creates a window to display these shapes using GLUT (OpenGL Utility Toolkit).

Experiment No. 3

Experiment Title: An OpenGL program to draw four stars

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(0.0, 800.0, 0.0, 800.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_POLYGON);
        glVertex2i(220, 260);
        glVertex2i(120, 300);
        glVertex2i(220, 340);
        glVertex2i(260, 440);
        glVertex2i(300, 340);
        glVertex2i(400, 300);
        glVertex2i(300, 260);
        glVertex2i(260, 160);
```

```
    glVertex2i(220, 260);
glEnd();

glColor3f(0, 1, 0);
glBegin(GL_POLYGON);
    glVertex2i(360, 400);
    glVertex2i(260, 440);
    glVertex2i(360, 480);
    glVertex2i(400, 580);
    glVertex2i(440, 480);
    glVertex2i(540, 450);
    glVertex2i(440, 400);
    glVertex2i(400, 300);
    glVertex2i(360, 400);
glEnd();

glColor3f(0, 0, 1);
glBegin(GL_POLYGON);
    glVertex2i(360, 120);
    glVertex2i(260, 160);
    glVertex2i(360, 200);
    glVertex2i(400, 300);
    glVertex2i(440, 200);
    glVertex2i(540, 160);
    glVertex2i(440, 120);
    glVertex2i(400, 20);
    glVertex2i(360, 120);
glEnd();

glColor3f(1, 1, 1);
glBegin(GL_POLYGON);
    glVertex2i(500, 260);
    glVertex2i(400, 300);
    glVertex2i(500, 340);
    glVertex2i(540, 450);
    glVertex2i(580, 340);
    glVertex2i(680, 300);
    glVertex2i(580, 260);
    glVertex2i(540, 160);
```

```
        glVertex2i(500, 260);
    glEnd();

    glFlush();

}

int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

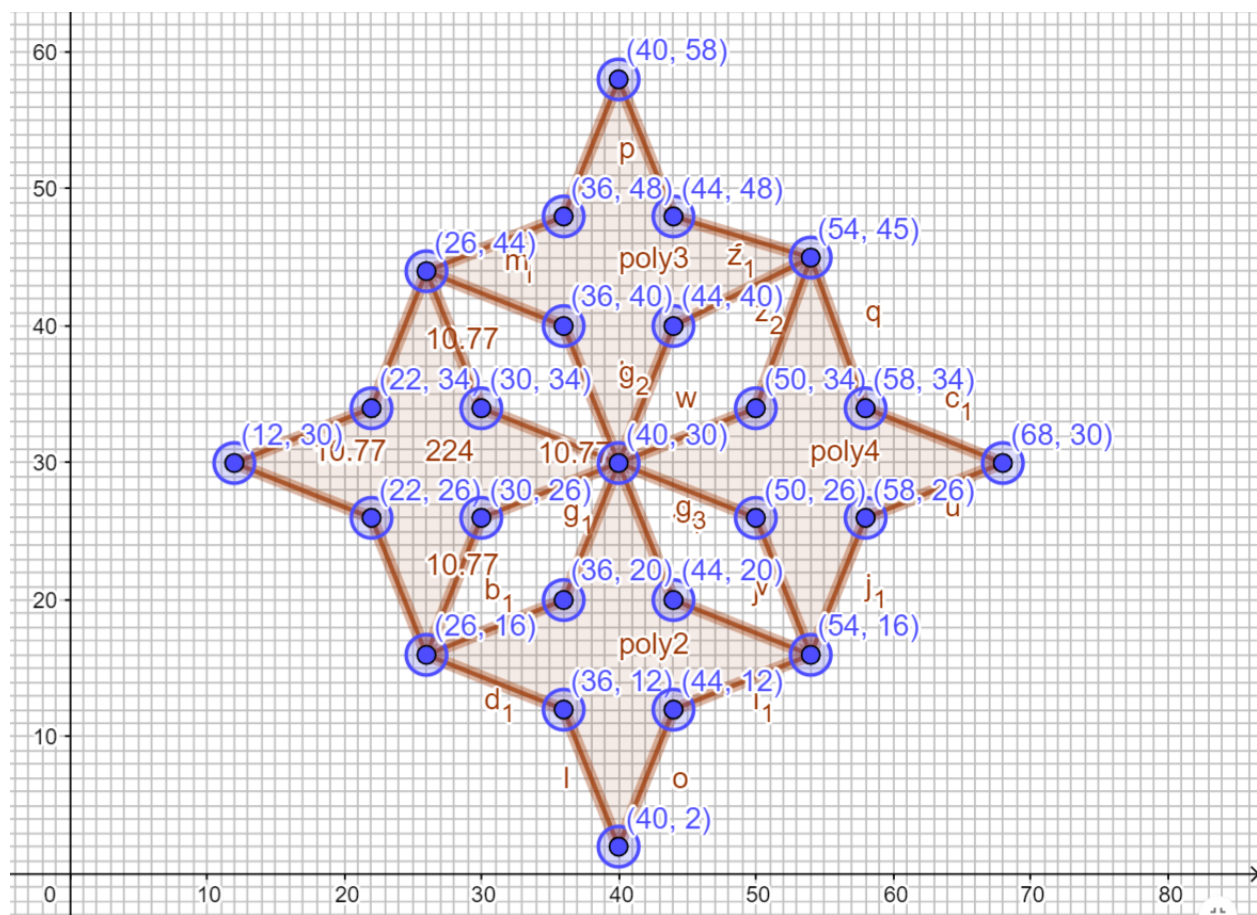
    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(900, 900);
    glutCreateWindow("Four Stars");

    init();
    glutDisplayFunc(drawShapes);

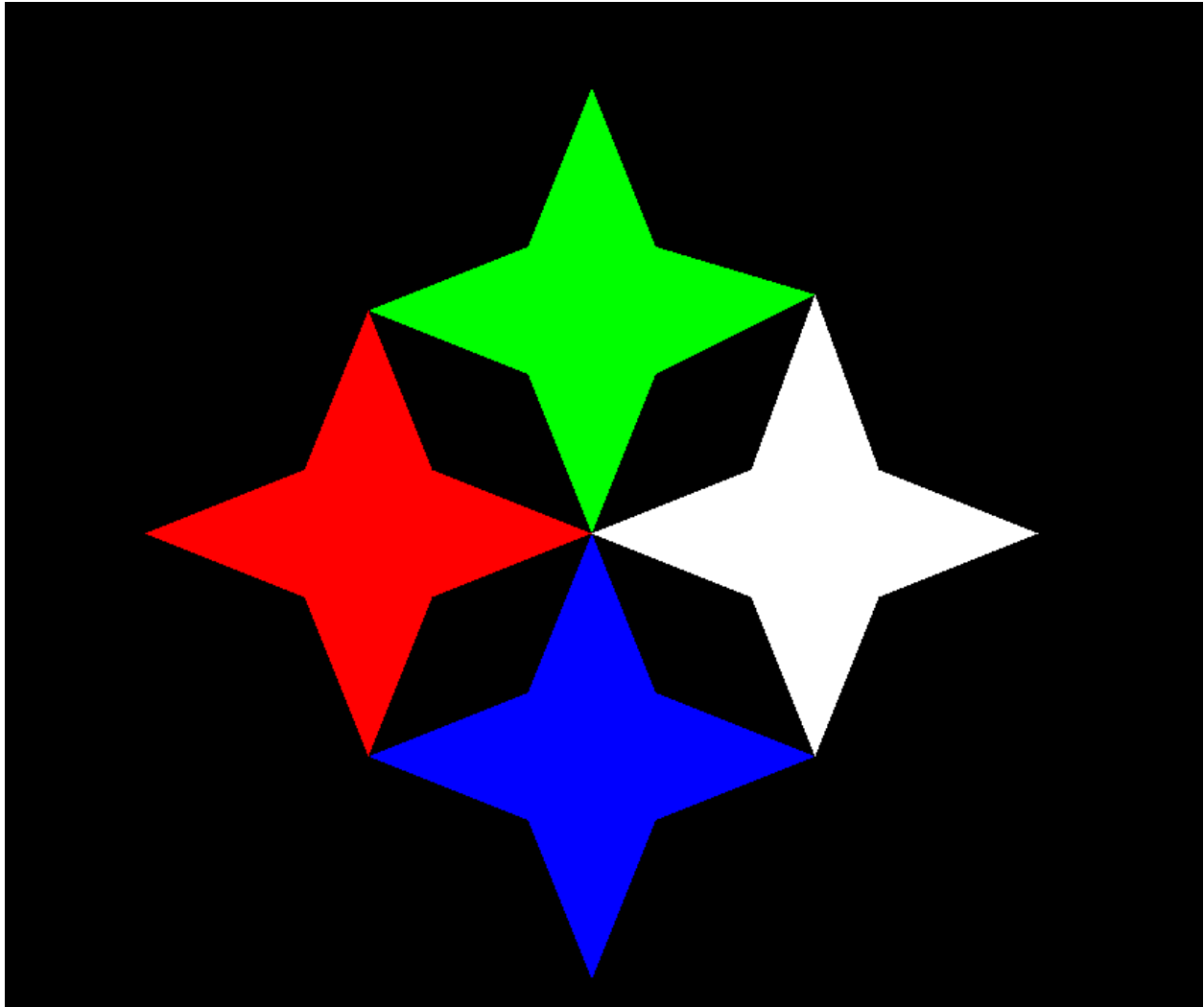
    glutMainLoop();

    return 0;
}
```

Graph:



Output: (Output data = graph data * 10)



Discussion:

This C++ program uses OpenGL to draw four colorful stars on a black background. The init function sets up the viewing area, and drawShapes specifies the coordinates for the vertices of each star. Different colors are achieved by changing the RGB values in the glColor3f function. The program creates a window to display these stars using GLUT (OpenGL Utility Toolkit).

Experiment No. 4

Experiment Title: An OpenGL program to draw chess board

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(0.0, 800.0, 0.0, 800.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    bool state = false;

    int box_size = 40;

    for(int i = 1; i <= 8; i++)
    {
        state = !state;
        for(int j = 1; j <= 8; j++)
        {
            if(state) glColor3f(0.0, 0.0, 0.0);
            else glColor3f(1.0, 1.0, 1.0);
        }
    }
}
```

```

        glBegin(GL_QUADS);
            glVertex2i(i * box_size, j * box_size);
            glVertex2i((i+1) * box_size, j * box_size);
            glVertex2i((i+1) * box_size, (j+1) * box_size);
            glVertex2i(i * box_size, (j+1) * box_size);
        glEnd();

        state = !state;
    }
}

glFlush();
}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(900, 900);
    glutCreateWindow("Chess Board");

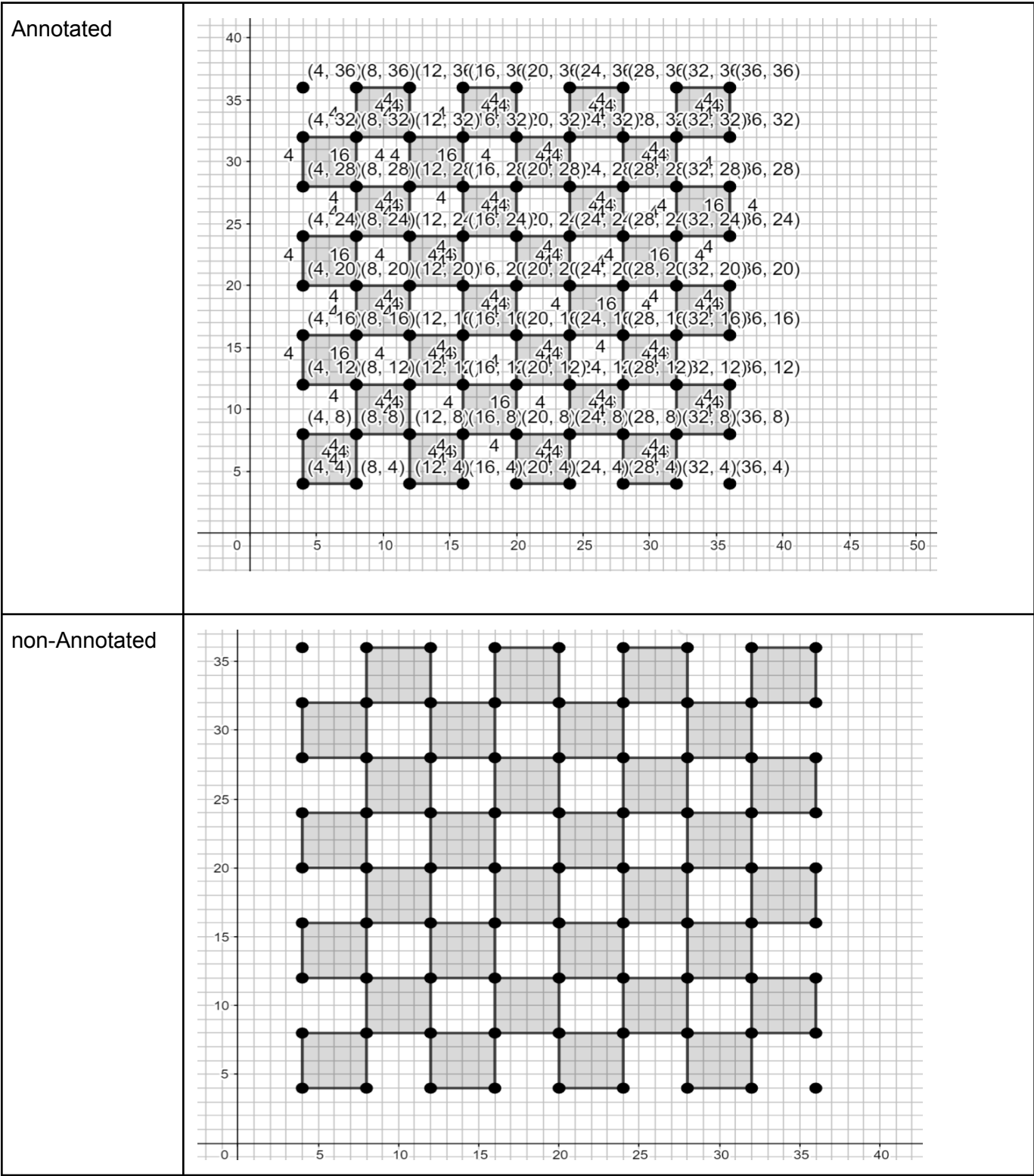
    init();
    glutDisplayFunc(drawShapes);

    glutMainLoop();

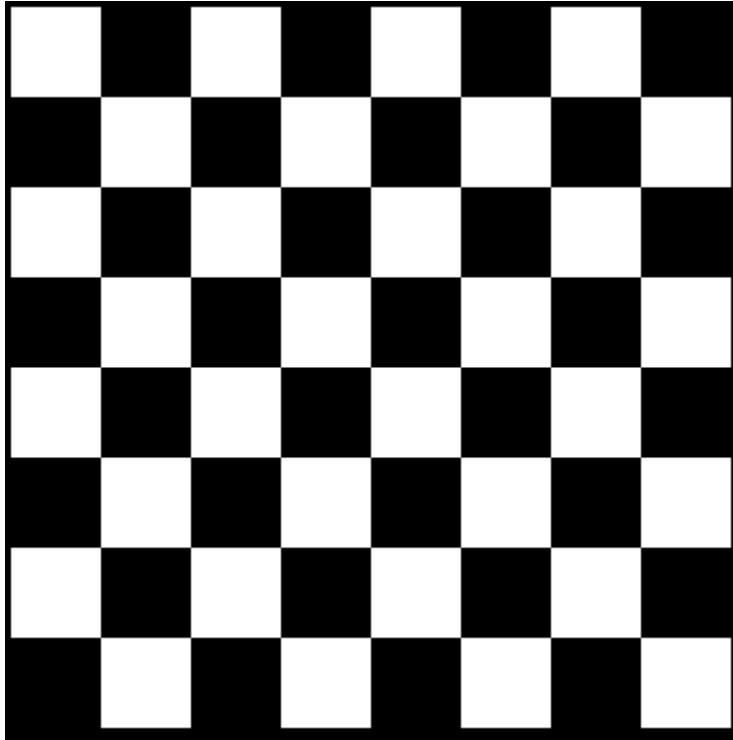
    return 0;
}

```


Graph:



Output:



Discussion:

This C++ program uses OpenGL to draw a simple chessboard pattern. The "drawShapes" function creates a chessboard grid by alternating black and white squares using nested loops. The size of each square is determined by the variable "box_size." The program showcases the use of OpenGL primitives, such as "glBegin" and "glVertex2i," to define and render the squares. The resulting output is a visually appealing chessboard displayed in a window.

Experiment No. 5

Experiment Title: An OpenGL program to draw a Line using Bresenham Line drawing algorithm

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int x1, y1, x2, y2;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(0.0, 800.0, 0.0, 800.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    int dx = x2 - x1;
    int dy = y2 - y1;

    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
        int p = (2*dy) - dx;

        for(int i = x1, j = y1; i <= x2, j <= y2;)
        {
```

```

        if(p < 0)
        {
            i++;
            if(i > x2 || j > y2) break;

            glVertex2f(i, j);
            p = p + (2*dy);
        }
        else
        {
            i++;
            j++;

            if(i > x2 || j > y2) break;

            glVertex2f(i, j);

            p = p + (2*dy) - (2*dx);
        }
    }
    glEnd();

glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(900, 900);
    glutCreateWindow("Bresenham Line drawing algorithm");

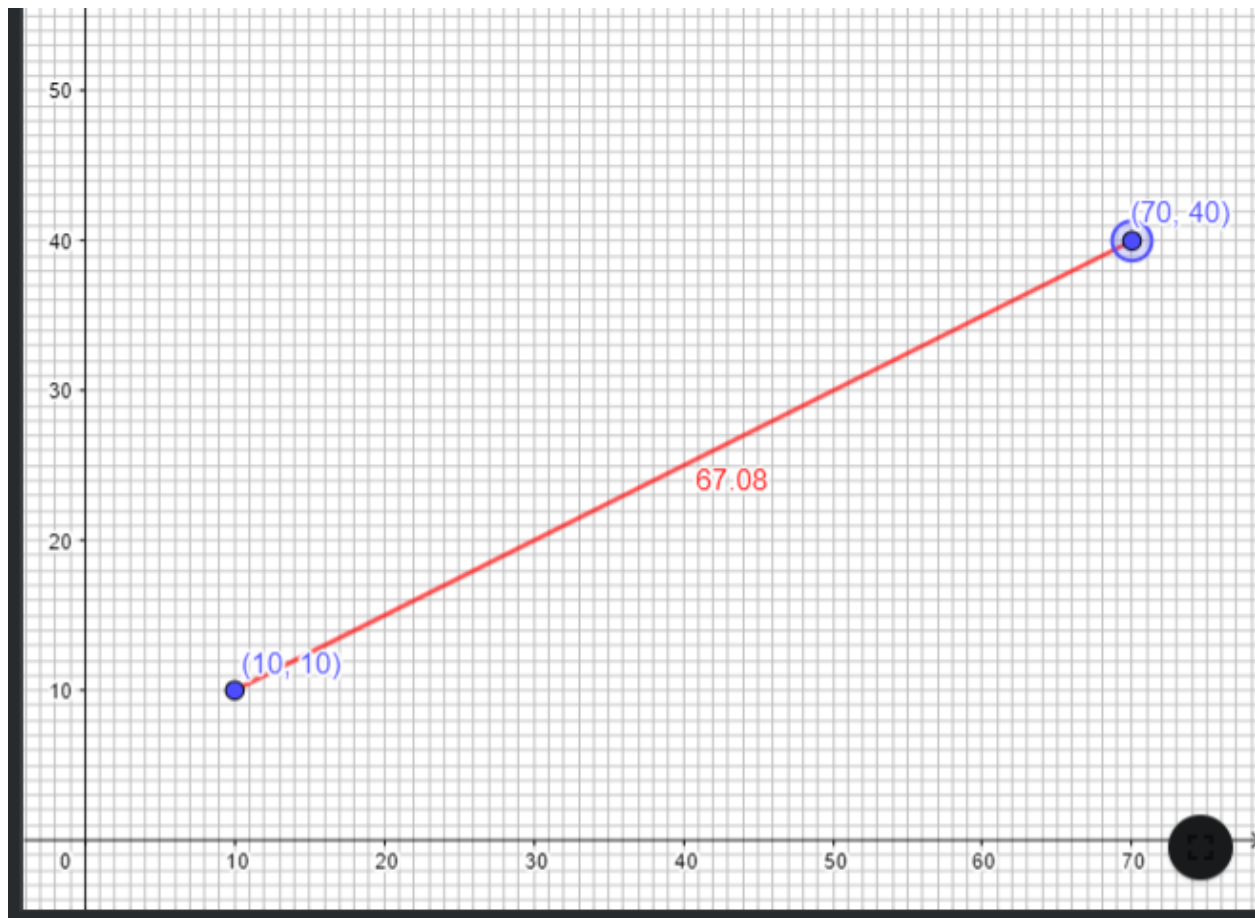
    init();
    glutDisplayFunc(drawShapes);

    cout << "Enter value of starting and end points " << endl;

```

```
    cout << "x1: ";  
    cin >> x1;  
    cout << endl;  
    cout << "y1: ";  
    cin >> y1;  
    cout << endl;  
  
    cout << "x2: ";  
    cin >> x2;  
    cout << endl;  
    cout << "y2: ";  
    cin >> y2;  
    cout << endl;  
  
    glutMainLoop();  
  
    return 0;  
}
```

Graph:

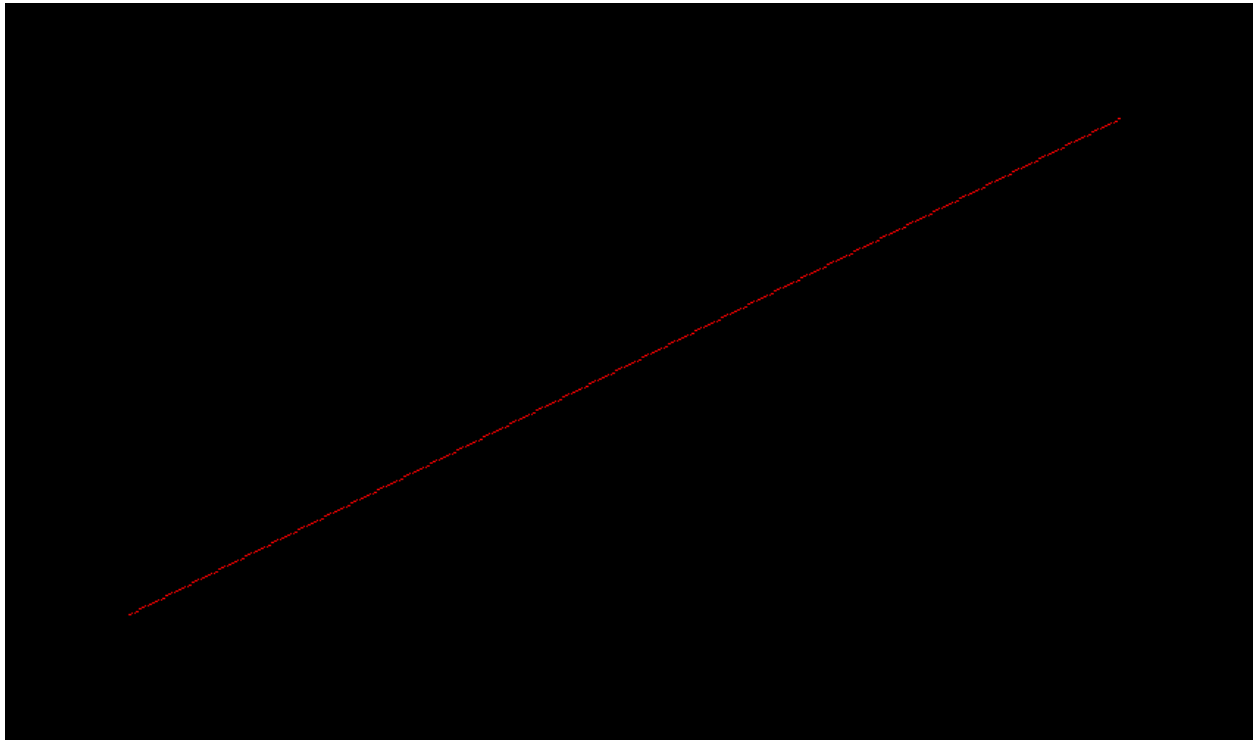


Input:

```
Enter value of starting and end points
x1: 100
y1: 100
x2: 700
y2: 400
```

Input data = graph data * 10

Output:



Discussion:

This C++ program uses OpenGL to draw a line using the Bresenham Line Drawing Algorithm. The user inputs the starting and ending points of the line, and the program efficiently calculates and displays the line by plotting pixels along the line path. The algorithm considers different cases based on the slope of the line and adjusts the pixel positions accordingly. The "glBegin" and "glVertex2f" functions are utilized to draw the line.

Experiment No. 6

Experiment Title: An OpenGL program to draw a circle using Midpoint circle algorithm

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int r;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-400.0, 400.0, -400.0, 400.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    int x = 0, y = r;

    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
        int p = 1-r;

        while(x <= y)
        {
            if(p < 0)
```



```

        {
            x++;

            p = p + (2*x) + 1;
        }
        else
        {
            x++;
            y--;

            p = p+(2*x)-(2*y);
        }
        glVertex2f(x,y);
        glVertex2f(y,x);
        glVertex2f(-x,-y);
        glVertex2f(-x,y);
        glVertex2f(x,-y);
        glVertex2f(y,-x);
        glVertex2f(-y,-x);
        glVertex2f(-y,x);
    }
    glEnd();

glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(900, 900);
    glutCreateWindow("Mid Point Circle Drawing Algorithm");

    init();
    glutDisplayFunc(drawShapes);

```

```
cout << "Enter RADIUS " << endl;

cout << "r: ";
cin >> r;

glutMainLoop();

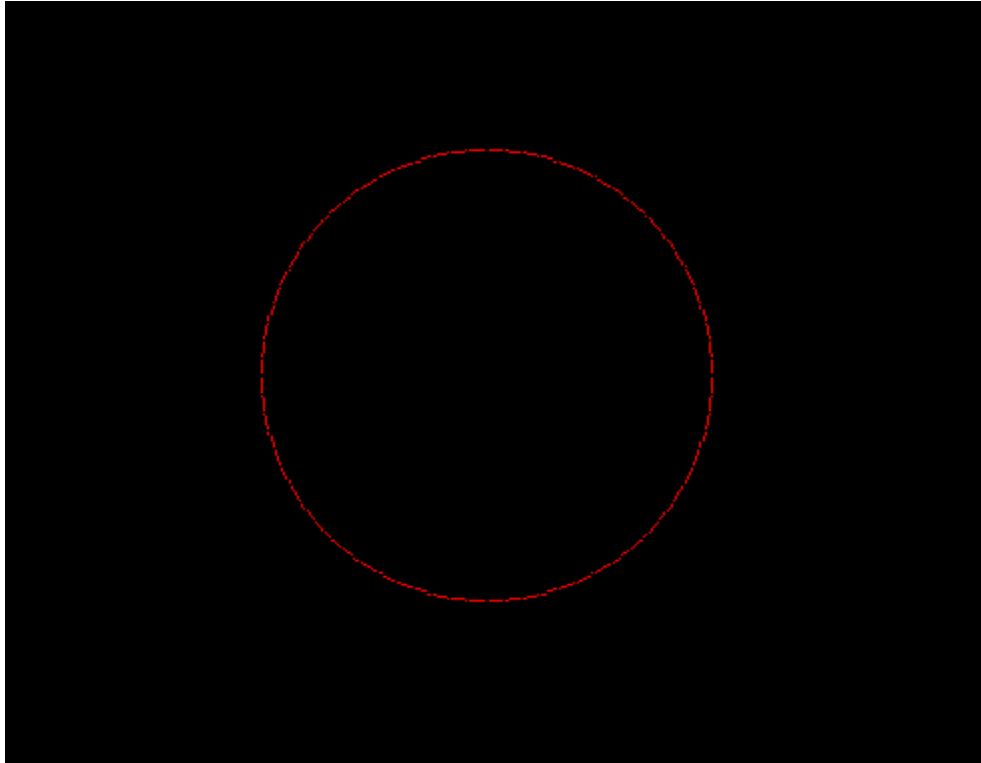
return 0;
}
```

Graph:

Input:

```
Enter Radius
r: 100
```

Output:



Discussion:

This C++ program uses OpenGL to draw a circle using the Midpoint Circle Drawing Algorithm. The user is prompted to input the radius of the circle, and the program then calculates and displays the circle using points based on the algorithm. The algorithm efficiently plots points on the circumference of the circle, creating a visually accurate representation. The "glBegin" and "glVertex2f" functions are used to draw the circle.

Experiment No. 7

Experiment: An OpenGL program to implement 2D transformations on a quads shape.

Task 1: Translation

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int ax, ay, bx, by, cx, cy, dx, dy, tx, ty;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();

    glColor3f(1, 1, 0);
```

```

        glBegin(GL_QUADS);
            glVertex2i(ax+tx, ay+ty);
            glVertex2i(bx+tx, by+ty);
            glVertex2i(cx+tx, cy+ty);
            glVertex2i(dx+tx, dy+ty);
        glEnd();

    glFlush();

}

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(100, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("Translation");

    init();
    glutDisplayFunc(drawShapes);

    cout << "Enter value for first shape " << endl;
    cout << "ax ";
    cin >> ax;
    cout << endl;
    cout << "ay ";
    cin >> ay;
    cout << endl;

    cout << "bx ";
    cin >> bx;
    cout << endl;
    cout << "by ";
    cin >> by;
    cout << endl;

    cout << "cx ";
    cin >> cx;

```

```
    cout << endl;
    cout << "cy ";
    cin >> cy;
    cout << endl;

    cout << "dx ";
    cin >> dx;
    cout << endl;
    cout << "dy ";
    cin >> dy;
    cout << endl;

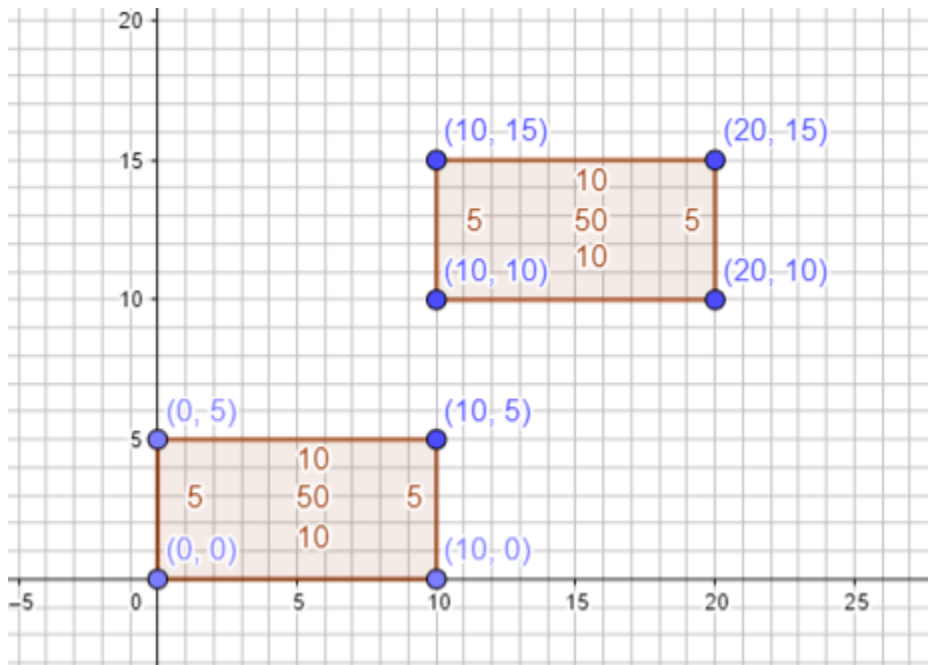
    cout << "Enter transform constants " << endl;
    cout << "tx ";
    cin >> tx;
    cout << endl;

    cout << "ty ";
    cin >> ty;
    cout << endl;

    glutMainLoop();

    return 0;
}
```

Graph:

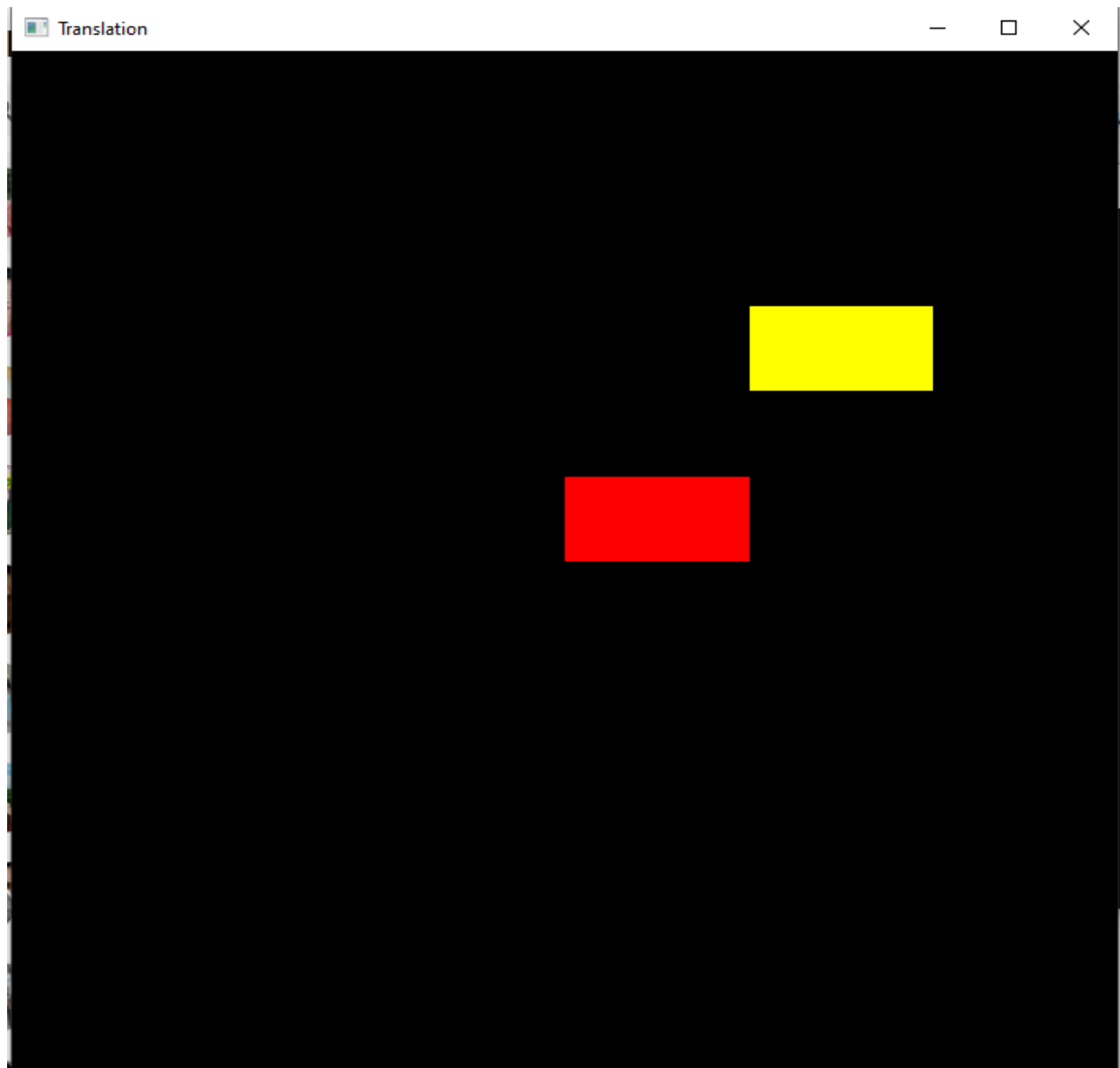


Input:

```
Enter value for first shape
ax 0
ay 0
bx 0
by 50
cx 100
cy 50
dx 100
dy 0
Enter transform constants
tx 100
ty 100
```

Input data = graph data * 10

Output:



Discussion:

This C++ program utilizes OpenGL to demonstrate translation transformation on a quadrilateral shape. The user inputs initial coordinates for the shape and translation constants for both the x and y directions. The program then translates the shape accordingly and displays both the original and translated shapes in different colors. Translation is achieved by adding the translation constants to each coordinate.

Task 2: Scaling

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int ax, ay, bx, by, cx, cy, dx, dy, sx, sy;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 1, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax*sx, ay*sy);
        glVertex2i(bx*sx, by*sy);
        glVertex2i(cx*sx, cy*sy);
        glVertex2i(dx*sx, dy*sy);
    glEnd();

    glColor3f(1, 0, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();
}
```

```
glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("Scalling");

    init();
    glutDisplayFunc(drawShapes);

    cout << "Enter value for first shape " << endl;
    cout << "ax ";
    cin >> ax;
    cout << endl;
    cout << "ay ";
    cin >> ay;
    cout << endl;

    cout << "bx ";
    cin >> bx;
    cout << endl;
    cout << "by ";
    cin >> by;
    cout << endl;

    cout << "cx ";
    cin >> cx;
    cout << endl;
    cout << "cy ";
    cin >> cy;
    cout << endl;
```

```

cout << "dx ";
cin >> dx;
cout << endl;
cout << "dy ";
cin >> dy;
cout << endl;

cout << "Enter scalling constants " << endl;
cout << "sx ";
cin >> sx;
cout << endl;

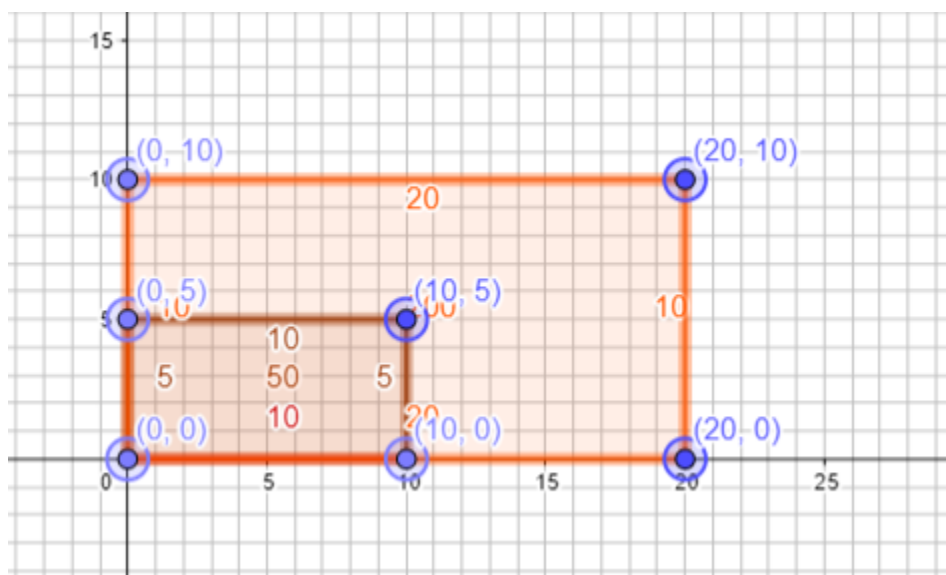
cout << "sy ";
cin >> sy;
cout << endl;

glutMainLoop();

return 0;
}

```

Graph:

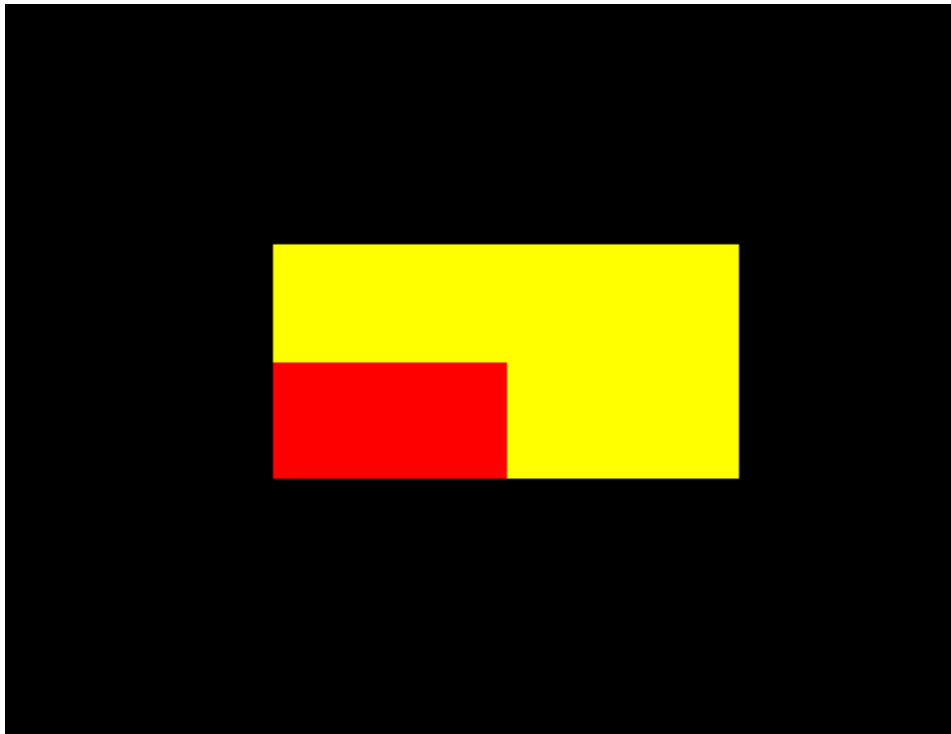


Input:

```
Enter value for first shape
ax 0
ay 0
bx 0
by 50
cx 100
cy 50
dx 100
dy 0
Enter scaling constants
sx 2
sy 2
```

Input data = graph data * 10

Output:



Discussion:

This C++ program uses OpenGL to demonstrate scaling transformation on a quadrilateral shape. The user inputs initial coordinates for the shape and scaling factors for both the x and y directions. The program then scales the shape accordingly and displays both the original and scaled shapes in different colors. Scaling is achieved by multiplying each coordinate by its respective scaling factor.

Task 3: Rotation

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
#include <cmath>
using namespace std;

int ax, ay, bx, by, cx, cy, dx, dy, theta;
#define PI acos(-1.0)

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();

    double r = PI*(theta)/180.0;
    double aX, aY, bX, bY, cX, cY, dX, dY;
    aX = ax*cos(r) - ay*sin(r);
    aY = ax*sin(r) + ay*cos(r);
```

```

    bX = bx*cos(r) - by*sin(r);
    bY = bx*sin(r) + by*cos(r);
    cX = cx*cos(r) - cy*sin(r);
    cY = cx*sin(r) + cy*cos(r);
    dX = dx*cos(r) - dy*sin(r);
    dY = dx*sin(r) + dy*cos(r);

    glColor3f(1, 1, 0);
    glBegin(GL_QUADS);
        glVertex2i(aX, aY);
        glVertex2i(bX, bY);
        glVertex2i(cX, cY);
        glVertex2i(dX, dY);
    glEnd();

glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("Rotation");

    init();
    glutDisplayFunc(drawShapes);

    cout << "Enter value for first shape " << endl;
    cout << "ax ";
    cin >> ax;
    cout << endl;
    cout << "ay ";
    cin >> ay;
    cout << endl;

```

```
cout << "bx ";
cin >> bx;
cout << endl;
cout << "by ";
cin >> by;
cout << endl;

cout << "cx ";
cin >> cx;
cout << endl;
cout << "cy ";
cin >> cy;
cout << endl;

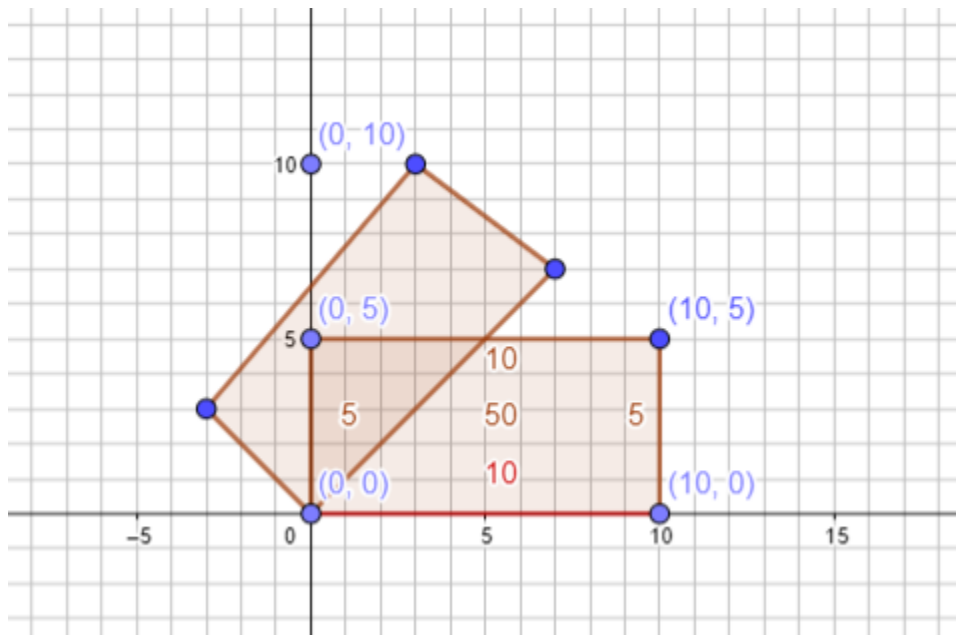
cout << "dx ";
cin >> dx;
cout << endl;
cout << "dy ";
cin >> dy;
cout << endl;

cout << "Enter rotation constant " << endl;
cout << "theta ";
cin >> theta;
cout << endl;

glutMainLoop();

return 0;
}
```


Graph:



Input:

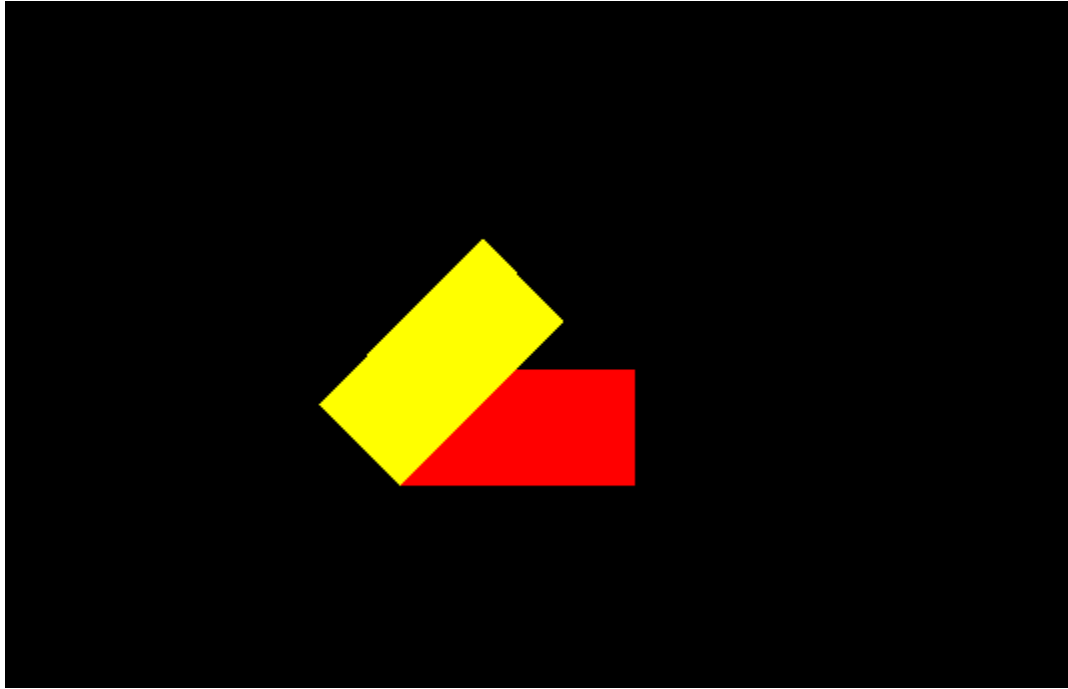
```
Enter value for first shape
ax 0
ay 0
bx 0
by 50
cx 100
cy 50
dx 100
dy 0

Enter rotation constant
theta 45
```

Input data = graph data * 10

Rotated data fits to nearest integer value in graph

Output:



Discussion:

This C++ program, using OpenGL, illustrates a rotation transformation applied to a quadrilateral shape. The user provides initial coordinates for the shape and specifies the rotation angle in degrees. The program then rotates the shape by the given angle and displays both the original and rotated shapes in different colors. Rotation is achieved using trigonometric functions to calculate the new coordinates after rotation.

Task 4: Reflection

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
#include <cmath>
using namespace std;

int ax, ay, bx, by, cx, cy, dx, dy;
char axis;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();

    if(axis == 'Y')
    {
        ax = ax*(-1);
        bx = bx*(-1);
    }
}
```

```

        cx = cx*(-1);
        dx = dx*(-1);
    }
    else
    {
        ay = ay*(-1);
        by = by*(-1);
        cy = cy*(-1);
        dy = dy*(-1);
    }

    glColor3f(1, 1, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();

glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("Reflaction");

    init();
    glutDisplayFunc(drawShapes);

    cout << "Enter value for first shape " << endl;
    cout << "ax ";
    cin >> ax;
    cout << endl;

```

```
    cout << "ay ";
    cin >> ay;
    cout << endl;

    cout << "bx ";
    cin >> bx;
    cout << endl;
    cout << "by ";
    cin >> by;
    cout << endl;

    cout << "cx ";
    cin >> cx;
    cout << endl;
    cout << "cy ";
    cin >> cy;
    cout << endl;

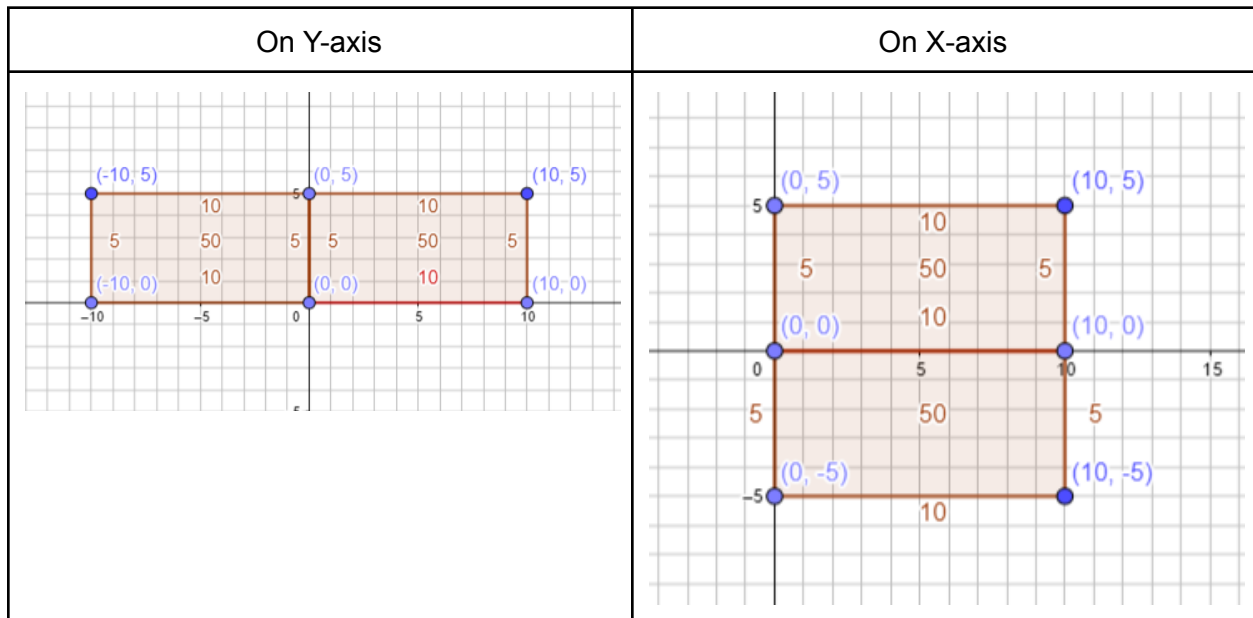
    cout << "dx ";
    cin >> dx;
    cout << endl;
    cout << "dy ";
    cin >> dy;
    cout << endl;

    cout << "Define axis (X/Y) " << endl;
    cout << "axis ";
    cin >> axis;
    cout << endl;

    glutMainLoop();

    return 0;
}
```

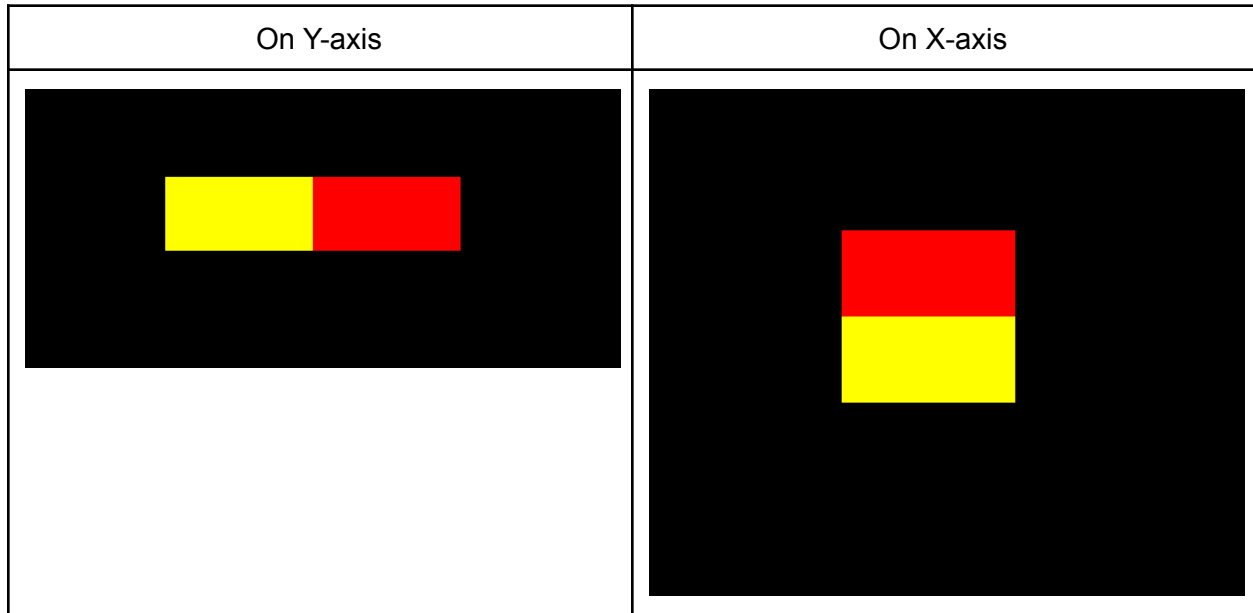
Graph:



Input:

On Y-axis	On X-axis
<pre> Enter value for first shape ax 0 ay 0 bx 0 by 50 cx 100 cy 50 dx 100 dy 0 Define axis (X/Y) axis Y </pre>	<pre> Enter value for first shape ax 0 ay 0 bx 0 by 50 cx 100 cy 50 dx 100 dy 0 Define axis (X/Y) axis X </pre>

Output:



Discussion:

This C++ program utilizes OpenGL to demonstrate reflection transformation on a quadrilateral shape. The user inputs the initial coordinates of the shape, and they specify the reflection axis (X or Y). The program then applies reflection to the shape and displays both the original and reflected shapes in different colors. Reflection is achieved by negating the coordinates along the specified axis.

Task 5: Shearing

Code:

```
#include<windows.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int ax, ay, bx, by, cx, cy, dx, dy, sh;
char axis;

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);

    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawShapes(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 0, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();

    if(axis == 'X')
    {
        ax = ax + ay*sh;
        bx = bx + by*sh;
        cx = cx + cy*sh;
```



```

        dx = dx + dy*sh;
    }
    else
    {
        ay = ay + ax*sh;
        by = by + bx*sh;
        cy = cy + cx*sh;
        dy = dy + dx*sh;
    }

    glColor3f(1, 1, 0);
    glBegin(GL_QUADS);
        glVertex2i(ax, ay);
        glVertex2i(bx, by);
        glVertex2i(cx, cy);
        glVertex2i(dx, dy);
    glEnd();

glFlush();

}
int main(int argc, char* argv[])
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);

    glutInitWindowPosition(1200, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("Shearing");

    init();
    glutDisplayFunc(drawShapes);

    cout << "Enter value for first shape " << endl;
    cout << "ax ";
    cin >> ax;
    cout << endl;
    cout << "ay ";

```

```
    cin >> ay;
    cout << endl;

    cout << "bx ";
    cin >> bx;
    cout << endl;
    cout << "by ";
    cin >> by;
    cout << endl;

    cout << "cx ";
    cin >> cx;
    cout << endl;
    cout << "cy ";
    cin >> cy;
    cout << endl;

    cout << "dx ";
    cin >> dx;
    cout << endl;
    cout << "dy ";
    cin >> dy;
    cout << endl;

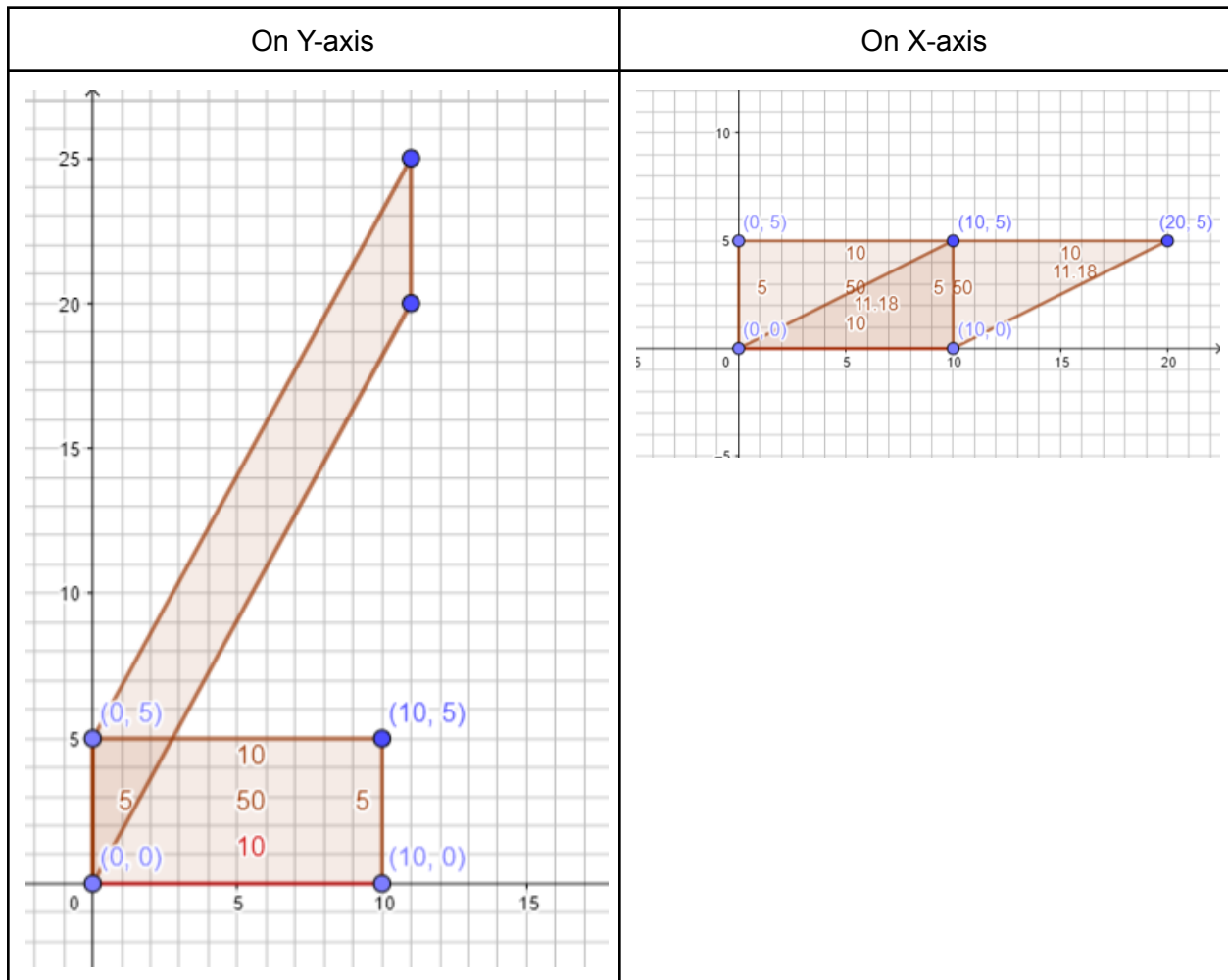
    cout << "Define axis (X/Y) " << endl;
    cout << "axis ";
    cin >> axis;
    cout << endl;

    cout << "Enter shearing constant " << endl;
    cout << "Value according to axis: ";
    cin >> sh;
    cout << endl;

    glutMainLoop();

    return 0;
}
```

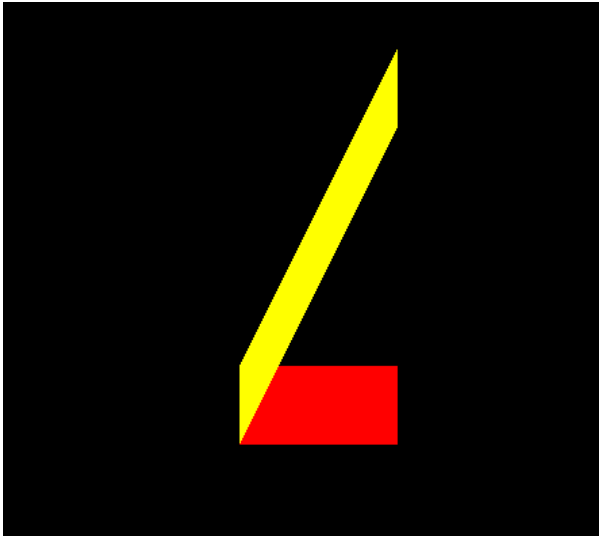
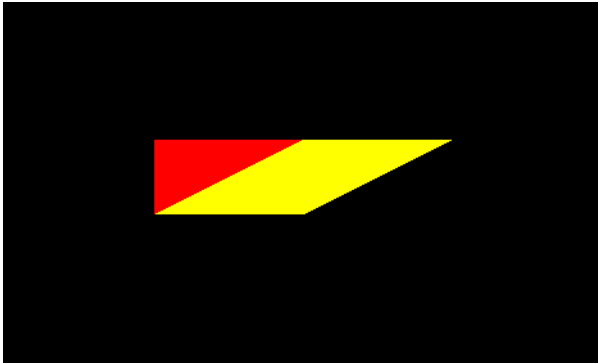
Graph:



Input:

On Y-axis	On X-axis
<pre>Enter value for first shape ax 0 ay 0 bx 0 by 50 cx 100 cy 50 dx 100 dy 0 Define axis (X/Y) axis Y Enter shearing constant Value according to axis: 2</pre>	<pre>Enter value for first shape ax 0 ay 0 bx 0 by 50 cx 100 cy 50 dx 100 dy 0 Define axis (X/Y) axis X Enter shearing constant Value according to axis: 2</pre>

Output:

On Y-axis	On X-axis
	

Discussion:

This C++ program uses OpenGL to demonstrate shearing transformation on a quadrilateral shape. The initial coordinates of the shape are taken as input, and the user specifies the shearing axis (X or Y) and the shearing constant. The program then applies shearing to the shape and displays both the original and sheared shapes in different colors. The shearing transformation is visualized by modifying the coordinates of the vertices based on the user's input.